1. API Documentation

Base URL --- http://localhost:3000

2.Endpoints

Method	**Endpoint**	**Description**	**Request Body Example**	**Response Example**
GET	/tasks	Retrieve all tasks	None	{ "tasks": [{ "id": 1, "title": "aaa", "description": "abababababababab" }, ...] }
GET	/tasks/:id	Retrieve a specific task by ID	None	{ "task": { "id": 1, "title": "aaa", "description": "abababababababab" } }
POST	/tasks	Create a new task	{ "title": "New Task", "description": "Description of new task" }	{ "message": "Task created", "task": { "id": 5, "title": "New Task", "description": "Description" }}
PUT	/tasks/:id	Update an existing task by ID	{ "title": "Updated Task", "description": "Updated description" }	{ "message": "Task updated", "task": { "id": 1, "title": "Updated Task", "description": "Updated" }}
DELETE	/tasks/:id	Delete a task by ID	None	{ "message": "Task deleted" }

3. Instructions for Running the API

1. **Install Node.js**
   - Download and install Node.js from https://nodejs.org.
2. **Save the Code**
   - Save the code to a file named index.js.
3. **Initialize the Project**

   ```
   npm init -y
   ```

4. **Install Dependencies**

   ```
   npm install express
   ```

5. **Run the Server**

   ```
   node server.js
   ```

6. **Test the API**
   - Use tools like **Postman**, **Thunder Client** or a browser for testing the API endpoints.

4. Report: Approach and Algorithm Choices

Approach

- The project uses **Node.js** and **Express.js** to build a RESTful API.
- Tasks are stored in memory using an array. This approach eliminates the need for a database and keeps the implementation lightweight.
- Basic validation ensures requests contain the necessary fields (title and description).

Algorithm Choices

1. **ID Management**
   - The id of each task is generated by incrementing the last task's id in the array.
   - This ensures unique IDs without a database.

2. **Validation**
    - Each endpoint checks for required fields or valid IDs.
    - If validation fails, appropriate status codes (400 or 404) and messages are returned.
3. **CRUD Operations**
    - The operations are designed to work on the in-memory array of tasks.
    - **Create:** Appends a new task to the array.
    - **Read:** Filters the array to find tasks or match IDs.
    - **Update:** Finds the task index by ID and replaces it.
    - **Delete:** Finds the task index by ID and removes it.